PARETO-FRONTIER-AWARE NEURAL ARCHITECTURE SEARCH

Anonymous authors
Paper under double-blind review

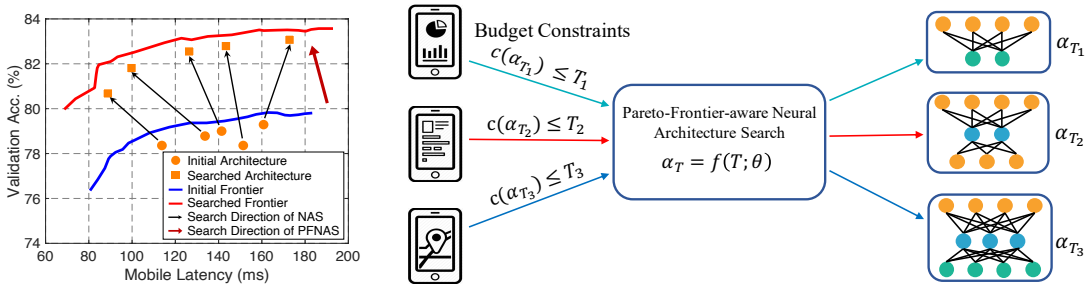
ABSTRACT

Designing feasible and effective architectures is essential for deploying deep models to real-world scenarios. In practice, one has to consider multiple objectives (*e.g.*, model performance and computational cost) and diverse constraints incurred by different computation resources. To address this, most methods seek to find promising architectures via optimizing a well pre-defined utility function. However, it is often non-trivial to design an ideal function that could well trade-off different objectives. More critically, in many real scenarios, even for the same platform, we may have different applications with various latency budgets. It would be fantastic if we can produce multiple promising architectures to fulfill each budget in the same search process. However, existing methods may have to perform an independent search for each budget, which is very inefficient and unnecessary. In this paper, we propose a Pareto-Frontier-aware Neural Architecture Search (PFNAS) method which seeks to learn the Pareto frontier (*i.e.*, the set of Pareto optimal architectures) *w.r.t.* multiple objectives. Here, we formulate the Pareto frontier learning problem as a Markov decision process (MDP). Relied on the MDP, we transform and absorb the objectives other than model performance into the constraints. To learn the whole Pareto frontier, we propose to find a set of Pareto optimal architectures which are uniformly distributed on the range of budget to form a frontier. Based on the learned frontier, we are able to easily find multiple promising architectures to fulfill all considered constraints in the same search process. Extensive experiments on three hardware platforms (*i.e.*, mobile, CPU, and GPU) show that the searched architectures by our PFNAS outperform the ones obtained by existing methods under different budgets.

1 INTRODUCTION

Deep neural networks (DNNs) (LeCun et al., 1989) have been the workhorse of many challenging tasks, including image classification (Krizhevsky et al., 2012; Srivastava et al., 2015) and semantic segmentation (Long et al., 2015; Noh et al., 2015). However, designing effective architectures is often labor-intensive and relies heavily on human expertise. To alleviate the computation burden of architecture design, neural architecture search (NAS) methods have been proposed to automatically design architectures (Zoph & Le, 2017; Liu et al., 2019). Existing studies show that the automatically discovered architectures often outperform the manually designed architectures in both image classification and language modeling tasks (Pham et al., 2018; Liu et al., 2019).

However, deep models often contain a large number of parameters and thus come with a very high computational cost. As a result, it is hard to deploy deep models to the hardware devices or application scenarios with limited computation resources. To obtain promising architectures that fulfill the computation constraint, we have to consider multiple kinds of objectives (*e.g.*, accuracy and computational cost). Thus, we seek to solve a Pareto optimization problem *w.r.t.* multiple objectives to perform architecture search (Tan et al., 2019). To solve this problem, one can design a utility function by computing a weighted sum/product of different objectives to find the desired architectures (Tan et al., 2019; Stamoulis et al., 2019). However, it is hard to design a utility function that could trade-off different kinds of objectives (Miettinen, 2012). As a result, the searched architectures do not necessarily satisfy the constraints (See results in Figure 3(b)). Thus, how to find feasible and effective architectures to satisfy the constraint becomes an important problem.



(a) Comparisons of search strategies between NAS and PFNAS.

(b) Model deployment under diverse budget constraints.

Figure 1: Illustration of the search strategy and applications of PFNAS. (a) Instead of finding a single Pareto optimal architecture, PFNAS seeks to learn the Pareto frontier. (b) PFNAS takes arbitrary budget constraint as input and output the architecture satisfied the budget constraint.

More critically, even for the same hardware platform, we may have different applications which result in diverse deployment budgets/requirements in terms of a specific objective. For example, a company may develop/maintain multiple applications on the same hardware device and each of them has a specific requirement of latency. To handle multiple application scenarios, we need to find a set of Pareto optimal architectures (*i.e.*, Pareto frontier) over multiple objectives. To this end, existing NAS methods may have to perform an independent search for each scenario (Tan et al., 2019), which is very inefficient yet unnecessary. To address this issue, we propose to directly learn the Pareto frontier over multiple objectives rather than find a single optimal architecture (See Figure 1(a)). Based on the learned Pareto frontier, one can easily find the desired architecture to fulfill an arbitrary budget. However, it is still unknown how to learn the Pareto frontier to produce effective architectures for multiple scenarios with diverse budgets.

In this paper, we propose a Pareto-Frontier-aware Neural Architecture Search (PFNAS) method to learn the Pareto frontier over two kinds of objectives. As shown in Figure 1(a), unlike existing methods that find the optimal architectures, we seek to learn the Pareto frontier (*i.e.*, improving the blue curve to the red curve). To this end, we formulate the Pareto frontier learning problem as a Markov decision process (MDP). Based on MDP, we transform and absorb the objectives other than model performance as the constraints and make decisions to find promising architectures satisfying them. To learn the whole Pareto frontier, we uniformly sampling budgets from its distribution and find a set of Pareto optimal architectures satisfying these budgets to form a frontier. Then, we exploit policy gradient to maximize the expected reward over different budgets. Based on the learned frontier, we may easily obtain the desired architectures given arbitrary budgets. To provide accurate reward, we propose an architecture evaluator to learn a Pareto dominance rule, which judges whether an architecture is better than another *w.r.t.* multiple objectives. By taking such a rule as the reward, we are able to iteratively find better frontiers during training. More critically, since our PFNAS exploits the shared knowledge across the search processes with multiple budgets, we find better architectures than those searched by an independent search for each budget (See results in Table 1).

We summarize the contributions of our paper as follows.

- We propose a Pareto-Frontier-aware Neural Architecture Search (PFNAS) method that simultaneously finds multiple Pareto optimal architectures (*i.e.*, Pareto frontier) over the whole range of computational cost (*e.g.*, latency). Based on the learned frontier, PFNAS takes arbitrary latency as the budget and automatically finds feasible architectures.
- We propose a Pareto dominance rule to judge whether an architecture is better than another under diverse computation budgets. By taking such a rule as the reward, our PFNAS is able to iteratively find better frontiers to approach the ground-truth Pareto frontier.
- Extensive experiments on three hardware platforms show the proposed method is able to find the architectures that not only satisfy diverse computation budgets but also outperform the architectures searched by existing methods.

2 RELATED WORK

Neural Architecture Search. Neural architecture search (NAS) has been proposed to automatically design effective architectures. Zoph & Le (2017) use reinforcement learning to discover the optimal configuration of each layer. Real et al. (2019) employ evolution algorithms and propose a new regularization method. Liu et al. (2019) propose DARTS, a differentiable NAS method by relaxing the search space to be continuous. However, these methods only search for the architectures with high accuracy but ignore the resource constraints of real-world applications.

Architecture Design with Resource Constraints. There is a growing interest in designing architectures under a resource constraint automatically. OFA (Cai et al., 2020) trains a powerful super network, from which we can directly get a specialized sub-network without additional training. Recently, PONAS (Huang & Chu, 2020) has been proposed to build an accuracy table to find architectures under a single constraint. However, given various resource budgets, these methods need to repeat the architecture search process for each budget. By contrast, our PFNAS only needs to search once to produce multiple architectures that satisfy diverse resource budgets simultaneously.

Pareto Frontier Learning. Pareto frontier learning aims to find a set of Pareto optimal solutions by solving a multi-objective optimization problem. Most methods convert the problem into a single-objective problem by constructing a weighted sum/product utility function (Wierzbicki, 1982; Miettinen, 2012). To simultaneously find multiple Pareto optimal solutions (*i.e.*, Pareto frontier), many methods exploit evolutionary algorithms (Deb et al., 2002; Kim et al., 2004) to perform a parallel search. Recently, some NAS methods aim to find a single Pareto optimal architecture by making a trade-off between accuracy and computational cost (Cheng et al., 2018; Dong et al., 2018). However, it is still unknown how to learn the Pareto frontier in NAS.

3 PROPOSED METHOD

3.1 PROBLEM DEFINITION

Notations. Let $\mathcal{T}$ be the distribution of discrete random variable $T \sim \mathcal{T}$, where T denotes the upper bound of any budget constraint, such as latency, the number of multiply-adds (MAdds) and memory consumption. Given an architecture space Ω and an architecture α, we use $c(\alpha)$ and $\mathrm{Acc}(\alpha)$ to measure the cost and the validation accuracy of α, respectively. We compute the reward of α using a function $R(\alpha; w)$ parameterized by w. Without loss of generality, we use $\alpha_T^{(i)}$ to denote the i-th searched architecture under the budget constraint $c(\alpha_T^{(i)}) \leq T$. We use $\mathbb{1}[A]$ to denote an indicator function, where $\mathbb{1}[A] = 1$ if A is true and $\mathbb{1}[A] = 0$ otherwise.

In this paper, we focus on the neural architecture search problem with multiple objectives (*e.g.*, the model performance and latency) and seek to find promising architectures under arbitrary constraints in the same search process. This problem, however, is non-trivial since it is hard to design a utility function to well trade-off the multiple kinds of objectives (Miettinen, 2012). Moreover, in real-world applications where we should consider diverse application scenarios with different budget constraints, performing an independent search for each scenario (Tan et al., 2019) would very inefficient yet unnecessary. To address these issues, we propose to learn the Pareto frontier (*i.e.*, the set of Pareto optimal architectures) *w.r.t.* different objectives instead of finding a single optimal architecture. Based on the learned frontier, it would be easy to select an architecture that fulfills arbitrary latency budgets.

3.2 PARETO-FRONTIER-AWARE NEURAL ARCHITECTURE SEARCH

In this paper, we propose a Pareto-Frontier-aware Neural Architecture Search (PFNAS) method that simultaneously finds multiple Pareto optimal architectures (*i.e.*, Pareto frontier). To this end, we formulate the optimization problem into a Markov decision process (MDP). Specifically, we transform and absorb the objectives other than model performance as a constraint and make decisions to find promising architectures to fulfill this budget. To cover the whole range of budgets, we uniformly sample different budgets and maximize the expected reward over all the decisions conditioned on them. Formally, a typical MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R)$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the state transition distribution, and $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. Here, we define the budget as a state, the decision to find an architecture

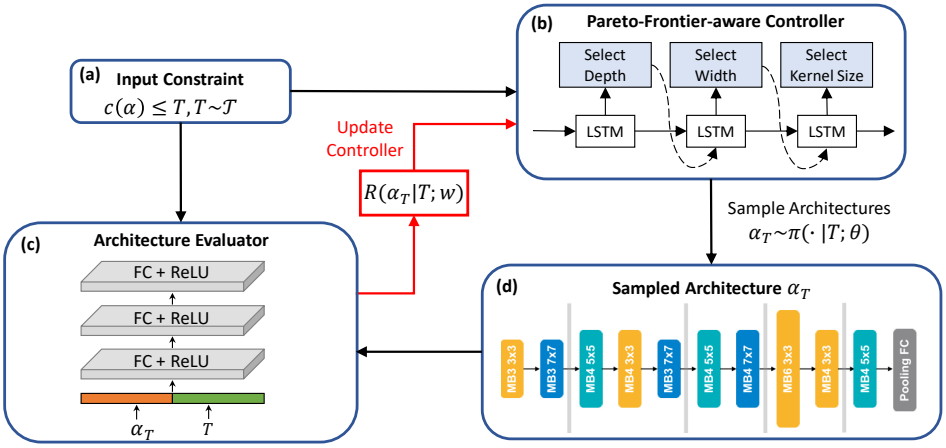


Figure 2: The overview of the proposed PFNAS. Our PFNAS takes a budget constraint as input and produces promising architectures that satisfy the budget constraints. The orange and green boxes in (c) denote the embeddings of the architecture α_T and the budget w.r.t. T.

satisfying any budget as an action. To find Pareto optimal (also called non-dominated) solutions, we develop a Pareto dominance rule to compute the reward (See Section 3.3). Here, we exploit the policy gradient method (Williams, 1992) to the MDP problem.

As shown in Figure 2, to find promising architectures under diverse budget constraints, we develop a conditional model that takes a budget T as input and outputs an architecture α_T satisfying the budget constraint $c(\alpha_T) \leq T$. Formally, the PFNAS model can be represented by $\alpha_T = f(T;\theta)$, where α_T denotes the architecture under this budget constraint and θ denotes the learnable parameters. Based on the searhed architecture α_T, we further feed it together with the considered budget T into an architecture evaluator to compute the reward. To illustrate our method, we first revisit the NAS problem with a single budget and then generalize it to the problem with diverse budgets. Note that it is non-trivial to directly find the optimal architecture (Zoph & Le, 2017). By contrast, one can first learn a policy $\pi(\cdot;\theta)$ and then conduct sampling from it to find promising architectures, *i.e.*, $\alpha \sim \pi(\cdot;\theta)$. Given a budget T, the optimization problem becomes

$$\max_{\theta} \ \mathbb{E}_{\alpha\sim\pi(\cdot;\theta)} \left[R\left(\alpha; w\right)\right], \ \text{s.t.} \ \ c(\alpha) \leq T. \tag{1}$$

Here, $\pi(\cdot;\theta)$ is the learned policy parameterized by θ, and $R(\alpha, w)$ is the reward function parameterized by w that measures the joint performance of both the accuracy and the latency of α. We use $\mathbb{E}_{\alpha\sim\pi(\cdot;\theta)}\left[\cdot\right]$ to denote the expectation over the searched architectures.

However, Problem (1) only focuses on one specific budget constraint. In fact, we seek to learn the Pareto frontier over the whole range of budgets (*e.g.*, latency). However, this problem is hard to solve since there exist infinite Pareto optimal architectures. To address this, one can learn the approximated Pareto frontier by finding a set of uniformly distributed Pareto optimal points (Grosan & Abraham, 2008). In this paper, we uniformly sample latencies as the budgets and maximize the expected reward over them. Thus, the optimization problem can be formulated as

$$\max_{\theta} \ \mathbb{E}_{T\sim\mathcal{T}} \left[\mathbb{E}_{\alpha_T\sim\pi(\cdot|T;\theta)} \left[R\left(\alpha_T|T; w\right)\right]\right], \ \text{s.t.} \ \ c(\alpha_T) \leq T, \ T \sim \mathcal{T}, \tag{2}$$

where $\mathbb{E}_{T\sim\mathcal{T}}\left[\cdot\right]$ denotes the expectation over the budget. Unlike Eqn. (1), $\pi(\cdot|T;\theta)$ is the learned policy conditioned on the budget of T. To find the architectures satisfying the budget constraint, we take T into account to compute the reward $R(\cdot|T;w)$. We will illustrate this in Section 3.3.

From Eqn. (2), we aim to improve the overall ability to find promising architectures under arbitrary latency budget, *i.e.*, learning the Pareto frontier. It is worth noting that simultaneously finding multiple Pareto optimal architectures would benefit the search process for each scenario with a specific latency constraint due to the shared knowledge across them (See results in Table 1). To be specific, if we find a good architecture *w.r.t.* one budget, we can slightly change the width or depth of some modules to obtain promising architectures that satisfy the adjacent budgets.

3.3 REWARD DESIGN FOR PARETO FRONTIER LEARNING

In this section, we propose a Pareto dominance reward to train PFNAS. Specifically, to obtain the Pareto optimal architectures, we have to find the Pareto improvement direction to iteratively find better architectures. Here, Pareto improvement is a situation where some objectives will increase and no objectives will decrease. This situation is also called Pareto dominance where the better solution dominates the worse one. In this sense, an architecture is defined to be Pareto optimal when it is not dominated by any architectures in the search space. Thus, since the Pareto frontier is the set of Pareto optimal architectures, the key challenge to Pareto frontier learning becomes how to find Pareto optimal architectures by judging whether an architecture dominates another architecture.

To address this, we define a Pareto dominance rule for the NAS problem. In practice, the quality of an architecture should depend on both the satisfaction of the budget and the accuracy. Specifically, given a specific budget T, a good architecture should be the one with the cost lower than or equal to T and with high accuracy. Motivated by this, we devise a rule to compare two architecture and judge which one is dominative. Given any two architectures β_1, β_2, 1) if $c(\beta_1) \leq T$ and $c(\beta_2) \leq T$, the architecture with higher accuracy is dominative; 2) if at least one architecture has the latency higher than T, the architecture with lower latency is dominative. Formally, we use a function $d(\cdot)$ to represent the above rule:

$$d\big(\beta_1, \beta_2, T\big) = \begin{cases} \mathbb{1}[\mathrm{Acc}(\beta_1) \geq \mathrm{Acc}(\beta_2)], & \text{if } \ c(\beta_1) \leq T \text{ and } c(\beta_2) \leq T; \\ \mathbb{1}[c(\beta_1) \leq c(\beta_2)], & \text{otherwise.} \end{cases} \tag{3}$$

Here, $d(\beta_1, \beta_2, T) = 1$ if β_1 dominates β_2 and $d(\beta_1, \beta_2, T) = 0$ otherwise. Similar rules are also found in the conventional constrained optimization problems (Deb et al., 2002). Note that Eqn. (3) can be considered as a hard threshold function which helps to guide the controller to find an architecture satisfying the budget constraints (See results in Sections 4).

From Eqn. (3), the Pareto dominance rule requires architecture pairs to find the Pareto optimal architectures. However, the controller model only finds an architecture at a time and thus Eqn. (3) cannot be directly used to compute the reward. To address this issue, we propose to train an architecture evaluator $R(\cdot|T; w)$ to learn the proposed Pareto dominance rule and output a scalar as the reward for the scenario with $c(\alpha) \leq T$. Since the proposed rules are built on the architecture comparisons, we train the architecture evaluator using a pairwise ranking loss, which has been widely used in ranking problems (Freund et al., 2003; Burges et al., 2005; Chen et al., 2009). Given M architectures, there are $M(M-1)$ architecture pairs in total after omitting the pairs with the same architecture. Assuming that there are K budgets, the pairwise ranking loss becomes

$$L(w) = \frac{1}{KM(M-1)} \sum_{k=1}^{K} \sum_{i=1}^{M} \sum_{j=1, j\neq i}^{M} \phi\big(\left(R(\beta_i|T_k; w) - R(\beta_j|T_k; w)\right) \cdot d(\beta_i, \beta_j, T_k)\big), \tag{4}$$

where $\phi(z) = \max(0, 1-z)$ is the hinge loss function. Due to the page limit, we put more discussions on Eqn. (4) in the supplementary.

Data preparation. We first train a super network with the progressive shrinking strategy. Then, we randomly sample M=16,000 architectures from the architecture space Ω and measure their accuracy $\mathrm{Acc}(\cdot)$ on 10,000 validation images sampled from the training set of ImageNet (Deng et al., 2009) using the super network. We also measure their latency $c(\cdot)$ on hardware devices. We record the results using a set of triplets $\{(\beta_i, c(\beta_i), \mathrm{Acc}(\beta_i))\}_{i=1}^{M}$.

3.4 TRAINING AND INFERENCE METHODS

As shown in Algorithm 1, we first train the super network using the progressive shrinking technique (Cai et al., 2020). Then, we sequentially train the architecture evaluator and the controller. We detail the training methods of the architecture evaluator and the controller below.

Learning the architecture evaluator $R(\alpha|T; w)$**.** To learn the Pareto frontier, we propose an architecture evaluator to compute the reward based on the proposed Pareto dominance rule. According to Eqn. (4), based on collected training data and $\{T_k\}_{k=1}^{K}$, the gradient w.r.t. w becomes

$$\nabla_w(w) = \frac{1}{KM(M-1)} \sum_{k=1}^{K} \sum_{i=1}^{M} \sum_{j=1, j\neq i}^{M} \nabla_w \phi\big(\left(R(\beta_i|T_k; w) - R(\beta_j|T_k; w)\right) \cdot d(\beta_i, \beta_j, T_k)\big). \tag{5}$$

Algorithm 1 Training method of PFNAS.

Require: Latency distribution $\mathcal{T}$, learning rate η, parameters M, N and K.
1: Initialize model parameters θ for the controller and w for the architecture evaluator.
2: *// Collect the architectures with the validation accuracy and the latency*
3: Train the super network on the training set with progressive shrinking strategy (Cai et al., 2020).
4: Randomly sample a set of architecture $\{\beta_i\}_{i=1}^{M}$ from the search space.
5: Measure the cost and the accuracy on the validation data set to construct set $\{(\beta_i, c(\beta_i), \mathrm{Acc}(\beta_i))\}_{i=1}^{M}$.
6: *// Train the architecture evaluator*
7: **while** not converge **do**
8: Sample a set of latencies $\{T_k\}_{k=1}^{K}$ from $\mathcal{T}$.
9: Update the architecture evaluator parameters w by descending the gradient:
10: $$w \leftarrow w - \eta \frac{1}{KM(M-1)} \sum_{k=1}^{K} \sum_{i=1}^{M} \sum_{j=1, j\neq i}^{M} \nabla_w \phi\big((R(\beta_i|T_k; w) - R(\beta_j|T_k; w)) \cdot d(\beta_i, \beta_j, T_k)\big).$$
11: **end while**
12: *// Train the controller*
13: **while** not converge **do**
14: Sample a set of latencies $\{T_k\}_{k=1}^{K}$ from $\mathcal{T}$.
15: Obtain $\{\alpha_{T_k}^{(i)}\}_{i=1}^{N}$ according to the policy $\pi(\cdot|T_k; \theta)$ for each $k \in \{1, \cdots, K\}$.
16: Update the controller parameters θ via policy gradient by ascending the gradient:
17: $$\theta \leftarrow \theta + \eta \frac{1}{KN} \sum_{k=1}^{K} \sum_{i=1}^{N} \left[\nabla_\theta \log \pi(\alpha_{T_k}^{(i)}|T_k; \theta) R(\alpha_{T_k}^{(i)}|T_k; w) + \lambda \nabla_\theta H(\pi(\cdot|T_k; \theta))\right].$$
18: **end while**

Learning the controller $f(T; \theta)$. The controller model $f(T; \theta)$ takes any given latency budget as input and outputs promising architectures to fulfill the budget. We learn the controller with policy gradient and use an entropy regularization term to encourage exploration. The objective becomes

$$J(\theta) = \mathbb{E}_{T\sim\mathcal{T}} \left[\mathbb{E}_{\alpha_T \sim \pi(\cdot|T;\theta)} \left[R\left(\alpha_T|T; w\right)\right] + \lambda H\big(\pi(\cdot|T; \theta)\big)\right], \tag{6}$$

where $H(\cdot)$ evaluates the entropy of the policy and λ is a hyper-parameter. In each iteration, we first sample $\{T_k\}_{k=1}^{K}$ from the distribution $\mathcal{T}$, and then sample N architectures $\{\alpha_{T_k}^{(i)}\}_{i=1}^{N}$ for each budget T_k. Thus, the gradient of Eqn. (6) w.r.t. θ becomes[1]

$$\nabla_\theta J(\theta) \approx \frac{1}{KN} \sum_{k=1}^{K} \sum_{i=1}^{N} \left[\nabla_\theta \log \pi(\alpha_{T_k}^{(i)}|T_k; \theta) R(\alpha_{T_k}^{(i)}|T_k; w) + \lambda \nabla_\theta H(\pi(\cdot|T_k; \theta))\right]. \tag{7}$$

Inferring architecture under diverse budgets. Based on the learned policy $\pi(\cdot|T; \theta)$, we conduct sampling to find promising architectures. Specifically, given any arbitrary latency T, we first sample several candidate architectures from $\pi(\cdot|T; \theta)$ and then select the architecture with the highest validation accuracy. Note that we train PFNAS using a finite number of discrete latencies. During inference, to enable T to be any value, we perform a linear interpolation between the embeddings of two adjacent discrete latencies to represent the considered latency (See more details in supplementary).

Training cost of PFNAS. The training of super network takes about 1.5 days based on 32 GPUs. The data preparation process takes about 40 GPU hours. We train the architecture evaluator for about 15 minutes. The training of the controller takes about 2 GPU hours, which is more efficient than the search process of most methods. Given K budgets, PFNAS only searches once and thus takes approximately $1/K$ search cost of the total cost to train a controller model for each budget.

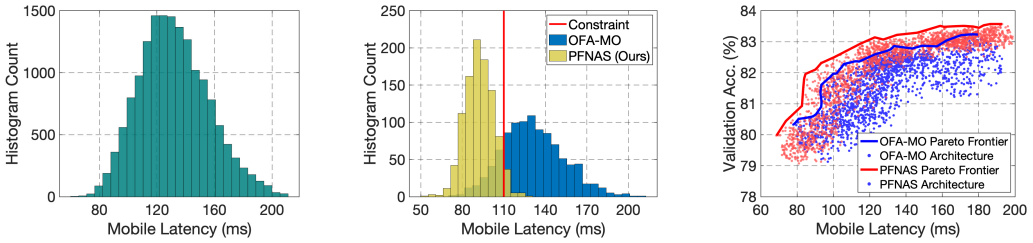
4 EXPERIMENTS

We apply PFNAS to search for architectures under diverse latency budgets on three kinds of hardware platforms, including mobile devices (Google Pixel1), CPU devices (Intel Core i5-7400), and GPU devices (NVIDIA TITAN X). For convenience, we use "Architecture-T" to represent the searched architecture that satisfies the latency budget w.r.t. T, *e.g.*, PFNAS-80. Due to the page limit, we put the visualizations of the searched architectures in the supplementary.

[1] We put the derivations of Eqn. (7) in the supplementary.

Table 1: Comparisons with state-of-the-art architectures on Google Pixel1 phone. * denotes the best architecture reported in the original paper. "-" denotes the results that are not reported. All the models are evaluated on 224×224 images of ImageNet.

Architecture	Latency (ms)	Top-1 Acc. (%)	Top-5 Acc. (%)	#Params (M)	#MAdds (M)
MobileNetV3-Large (0.75×) (Howard et al., 2019)	93.0	73.3	-	4.0	155
MobileNetV2 (1.0×) (Sandler et al., 2018)	90.3	72.0	-	3.4	300
OFA-S-80	76.8	76.8	93.3	6.1	350
OFA-MO-80	77.6	76.6	93.2	7.9	340
PFNAS-80 (Ours)	79.9	**77.5**	**93.7**	7.3	349
ProxylessNAS-Mobile (Cai et al., 2019)	97.7	74.6	-	4.1	319
MobileNetV3-Large (1.0×) (Howard et al., 2019)	107.7	75.2	-	5.4	219
OFA (Cai et al., 2020)	109.3	78.1	94.0	8.2	354
OFA-S-110	109.2	77.5	93.6	6.4	406
OFA-MO-110	106.3	78.0	93.8	8.4	478
PFNAS-110 (Ours)	106.8	**78.4**	**94.2**	9.9	451
MnasNet-A1 (1.0×) (Tan et al., 2019)	120.7	75.2	92.5	3.4	300
FBNet-C (Wu et al., 2019)	135.2	74.9	-	5.5	375
OFA (Cai et al., 2020)	133.7	78.4	94.1	8.4	388
OFA-S-140	130.0	77.7	93.7	6.6	428
OFA-MO-140	139.0	78.4	94.0	9.5	486
PFNAS-140 (Ours)	127.8	**78.7**	**94.3**	9.2	492
PONAS-C (Huang & Chu, 2020)	145.1	75.2	-	5.6	376
OFA* (Cai et al., 2020)	150.9	78.9	94.4	9.1	511
OFA-S-170	163.6	78.2	94.1	7.8	534
OFA-MO-170	165.0	78.8	94.4	8.5	584
PFNAS-170 (Ours)	167.1	**79.0**	**94.5**	10.0	606
DARTS (Liu et al., 2019)	176.6	73.1	91.0	4.7	574
PC-DARTS (Xu et al., 2020)	194.1	75.8	92.7	5.3	597
MnasNet-A1 (1.4×) (Tan et al., 2019)	205.5	77.2	93.5	6.1	592
EfficientNet B0 (Tan & Le, 2019)	237.7	77.3	93.5	5.3	390
OFA-S-200	197.5	78.3	94.2	8.4	629
OFA-MO-200	187.4	78.9	94.4	9.1	630
PFNAS-200 (Ours)	193.9	**79.2**	**94.7**	10.4	724



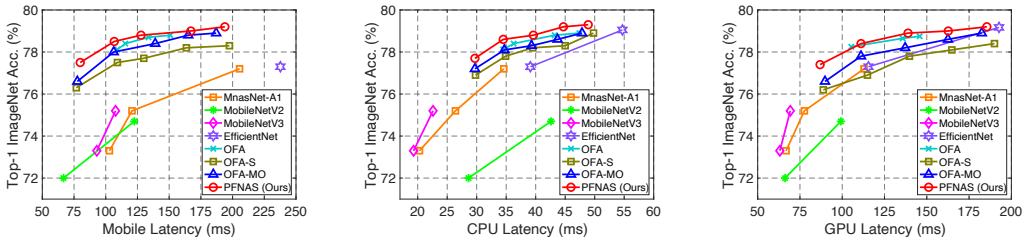
(a) Ground-truth latency histogram. (b) Comparisons of search results. (c) Comparisons of Pareto frontiers.

Figure 3: Latency histograms and Pareto curves of the architectures on mobile devices. (a) Ground-truth latency histogram of $16,000$ architectures that are uniformly sampled from the search space. (b) The latency histogram of $1,000$ architectures sampled by different methods given T=110ms. (c) The Pareto frontier of the architectures sampled by different methods.

Implementation details. We use MobileNetV3 (Howard et al., 2019) as the backbone to build the search space (Cai et al., 2020). We first obtain the range of latency by randomly sampling M=$16,000$ architectures from the search space (See Figure 3(a)). Then, we select K=5 latency budgets by evenly dividing the range (*e.g.*, $\{80, 110, 140, 170, 200\}$ on mobile devices). We put the discussion on the impact of K on the search performance of PFNAS and more implementation details in the supplementary materials.

4.1 Comparisons on hardware devices

We compare PFNAS with state-of-the-art methods on Google Pixel1 phone. We also consider the following baselines: 1) **OFA-MO** conducts architecture search based on OFA super network by exploiting the multi-objective reward (Tan et al., 2019). 2) **OFA-S** finds the best one from $16,000$ architectures sampled from the learned super network. From Table 1 and Figure 4(a), our PFNAS consistently achieves higher accuracy than other methods. Moreover, compared with the methods searching for different constrained architectures independently (*e.g.*, OFA and OFA-MO), our PFNAS only needs to search once to find the Pareto frontier instead of a single Pareto optimal solution. The



(a) Results on Google Pixel1. (b) Results on Intel Core i5 CPU. (c) Results on TITAN X GPU.

Figure 4: Comparisons of the architectures obtained by different methods on three hardware devices.

Table 2: Comparisons with different reward functions and search strategies on ImageNet. $P_s(\%)$ denotes the proportion of the searched architectures that satisfy the corresponding budget.

Reward	Pareto Frontier Learning	T=80ms		T=110ms		T=140ms		T=170ms		T=200ms	
		Acc.	P_s	Acc.	P_s	Acc.	P_s	Acc.	P_s	Acc.	P_s
Multi-objective Reward (Tan et al., 2019)	✗	76.6	1.4	78.0	33.2	78.4	54.0	78.8	90.5	78.9	99.9
	✓	77.0	3.2	78.1	43.9	78.5	88.4	78.9	99.1	78.9	99.9
Pareto Dominance Reward	✗	76.3	92.8	78.1	92.5	78.6	93.2	78.9	94.3	79.0	99.9
	✓	**77.2**	30.2	**78.4**	76.9	**78.7**	82.5	**79.0**	88.1	**79.2**	99.5

learned Pareto frontier benefits from the shared knowledge across the search process under different budgets, helping the decision makers to further select their preferred architectures.

We also visualize the latency histograms of the architectures searched on mobile devices in Figure 3(b). Given a latency budget of 110ms, only a few architectures produced by OFA-MO satisfy the budget, which demonstrates the multi-objective reward is hard to design to obtain the preferred architectures. By contrast, PFNAS uses the Pareto dominance reward to encourage the architectures to satisfy the desired budget constraints. Moreover, we compare the searched frontiers of different methods. Here, we combine the architectures searched by 5 independent OFA-MO runs under different budgets and select all the best architectures to generate an entire Pareto frontier. From Figure 3(c), our PFNAS finds a better frontier than OFA-MO due to the shared knowledge across the search process under different budgets. We also evaluate PFNAS on Intel Core i5-7400 CPU and NVIDIA TITAN X GPU. From Figure 4, PFNAS consistently find better architectures than existing methods for each latency budget on all considered devices (See more results in supplementary).

4.2 Ablation Studies of the Proposed Method

In this experiment, we investigate the effectiveness of the Pareto frontier search strategy and the Pareto dominance reward. From Table 2, the Pareto frontier search strategy tends to find better than the independent search process due to the shared knowledge across the search processes under different budgets. Moreover, compared with multi-objective reward, the Pareto dominance reward encourages the controller to find more architectures that satisfy the considered budget constraints. For example, even if only a few architectures have the latency lower than 80ms (See Figure 3(a)), we still achieve the P_s of 92.8% with the Pareto dominance reward. With both the Pareto frontier search strategy and the Pareto dominance reward, we yield the best search results under all budgets.

5 Conclusion

In this paper, we have proposed a novel Pareto-Frontier-aware Neural Architecture Search (PFNAS) method to find the Pareto frontier under diverse computation budget (*i.e.*, latency). Specifically, we train the PFNAS model to learn the Pareto frontier by maximizing the expected reward over a set of budgets. To provide accurate rewards under diverse budgets, we propose a Pareto dominance rule to judge whether an architecture is better than another and devise an architecture evaluator to learn this rule. In this way, PFNAS is able to learn the Pareto frontier and find promising architectures under diverse budgets. Extensive experiments on three platforms (*i.e.*, mobile, CPU, and GPU devices) demonstrate the effectiveness of the proposed method.

REFERENCES

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *International Conference on Machine Learning*, pp. 89–96, 2005.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

Han Cai, Chuang Gan, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020.

Wei Chen, Tie-Yan Liu, Yanyan Lan, Zhi-Ming Ma, and Hang Li. Ranking measures and loss functions in learning to rank. In *Advances in Neural Information Processing Systems*, pp. 315–323, 2009.

An-Chieh Cheng, Jin-Dong Dong, Chi-Hung Hsu, Shu-Huan Chang, Min Sun, Shih-Chieh Chang, Jia-Yu Pan, Yu-Ting Chen, Wei Wei, and Da-Cheng Juan. Searching toward pareto-optimal device-aware neural architectures. In *Proceedings of the International Conference on Computer-Aided Design*, pp. 1–7, 2018.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. In *Proceedings of the European Conference on Computer Vision*, pp. 517–531, 2018.

Yoav Freund, Raj Iyer, Robert E Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4(Nov):933–969, 2003.

Crina Grosan and Ajith Abraham. Generating uniformly distributed pareto optimal points for constrained and unconstrained multicriteria optimization. *INFOS*, pp. 27–29, 2008.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1314–1324, 2019.

Sian-Yao Huang and Wei-Ta Chu. Ponas: Progressive one-shot neural architecture search for very efficient deployment. *arXiv preprint arXiv:2003.05112*, 2020.

Mifa Kim, Tomoyuki Hiroyasu, Mitsunori Miki, and Shinya Watanabe. SPEA2+: improving the performance of the strength pareto evolutionary algorithm 2. In *Parallel Problem Solving from Nature*, volume 3242, pp. 742–751. Springer, 2004.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3431–3440, 2015.

Kaisa Miettinen. *Nonlinear Multiobjective Optimization*, volume 12. Springer Science & Business Media, 2012.

Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1520–1528, 2015.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*, pp. 4095–4104, 2018.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence*, volume 33, pp. 4780–4789, 2019.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in Neural Information Processing Systems*, pp. 2377–2385, 2015.

Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 481–497. Springer, 2019.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pp. 6105–6114, 2019.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828, 2019.

Andrzej P Wierzbicki. A mathematical basis for satisficing decision making. *Mathematical modelling*, 3(5):391–405, 1982.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10734–10742, 2019.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient differentiable architecture search. In *International Conference on Learning Representations*, 2020.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.